



# PROGRAMACIÓN II – UNIDAD 1

Ing. Gastón Weingand ([gaston.weingand@uai.edu.ar](mailto:gaston.weingand@uai.edu.ar))

# 5-Lenguaje python

## Python como calculadora

```
In [5]: 2 * 5 + 6 - (100 + 3)
```

```
Out[5]: -87
```

```
In [6]: 8/2, 8/4
```

```
Out[6]: (4.0, 2.0)
```

```
In [7]: 7 // 2, 7 % 2
```

```
Out[7]: (3, 1)
```

```
In [8]: 2 ** 10
```

```
Out[8]: 1024
```

# 5-Lenguaje python

## HOLA MUNDO Y COMENTARIOS

En un archivo llamado **holamundo.py** escribimos...

```
print("Hola mundo") #Comentario de línea
```

```
"""Comentario multi-  
línea"""
```

# 5-Lenguaje python

En Python los tipos básicos se dividen en:

- Números, como pueden ser 3 (entero), 15.57 (de coma flotante) o  $7 + 5j$  (complejos)
- Cadenas de texto, como "Hola Mundo"
- Valores booleanos: True (cierto) y False (falso).

```
# esto es una cadena  
c = "Hola Mundo"
```

```
# y esto es un entero  
e = 23
```

```
# podemos comprobarlo con la función type  
type(c)  
type(e)
```

# 5-Lenguaje python

## Operadores aritméticos

Operador	Descripción	Ejemplo
+	Suma	<code>r = 3 + 2</code> # r es 5
-	Resta	<code>r = 4 - 7</code> # r es -3
-	Negación	<code>r = -7</code> # r es -7
*	Multiplicación	<code>r = 2 * 6</code> # r es 12
**	Exponente	<code>r = 2 ** 6</code> # r es 64
/	División	<code>r = 3.5 / 2</code> # r es 1.75
//	División entera	<code>r = 3.5 // 2</code> # r es 1.0
%	Módulo	<code>r = 7 % 2</code> # r es 1

# 5-Lenguaje python

## Operadores a nivel de bit

Operador	Descripción	Ejemplo
&	and	<code>r = 3 &amp; 2 # r es 2</code>
	or	<code>r = 3   2 # r es 3</code>
^	xor	<code>r = 3 ^ 2 # r es 1</code>
~	not	<code>r = ~3 # r es -4</code>
<<	Desplazamiento izq.	<code>r = 3 &lt;&lt; 1 # r es 6</code>
>>	Desplazamiento der.	<code>r = 3 &gt;&gt; 1 # r es 1</code>

# 5-Lenguaje python

## Cadenas

```
a = "uno"
```

```
b = "dos"
```

```
c = a + b # c es "unodos"
```

```
c = a * 3 # c es "unounouno"
```

# 5-Lenguaje python

## Booleanos

Estos son los distintos tipos de operadores con los que podemos trabajar con valores booleanos, los llamados operadores lógicos o condicionales:

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	<code>r = True and False # r es False</code>
or	¿se cumple a o b?	<code>r = True or False # r es True</code>
not	No a	<code>r = not True # r es False</code>



# 5-Lenguaje python

## Booleanos

Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores):

Operador	Descripción	Ejemplo
<code>==</code>	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
<code>!=</code>	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<code>&lt;</code>	¿es a menor que b?	<code>r = 5 &lt; 3 # r es False</code>
<code>&gt;</code>	¿es a mayor que b?	<code>r = 5 &gt; 3 # r es True</code>

# 5-Lenguaje python

## Conversión de tipos primitivos

- `int (x)` Convierte x en un entero
- `long (x)` Convierte x en un entero largo
- `float (x)` Convierte x en un número de punto flotante
- `str (x)` Convierte x a una cadena. x puede ser del tipo float, entero o largo
- `hex (x)` Convierte x entero en una cadena hexadecimal
- `chr (x)` Convierte x entero a un caracter
- `ord (x)` Convierte el carácter x en un entero

El código

# 5-Lenguaje python

## Formateo de salida

```
a = 8
b= "Hello"
c= True

#Formateo de salida...
print("Concatenando " + str(a) + " " + b + " " + str(c)) #Obligatorio hacer cast a string
print("Concatenando",a,b,c)
print("Concatenando %s %s %s" % (a,b,c))

my_string = "Concatenando {} {} {}"
print(my_string.format(a,b,c))

print(f"Concatenando {a} {b} {c} ") #Sugerida...
```

# 5-Lenguaje python

## Variables y asignaciones

```
In [9]: a = 8  
        b = 2 * a  
        b
```

Out[9]: 16

```
In [10]: a + b
```

Out[10]: 24

```
In [11]: a = a + 1  
        a
```

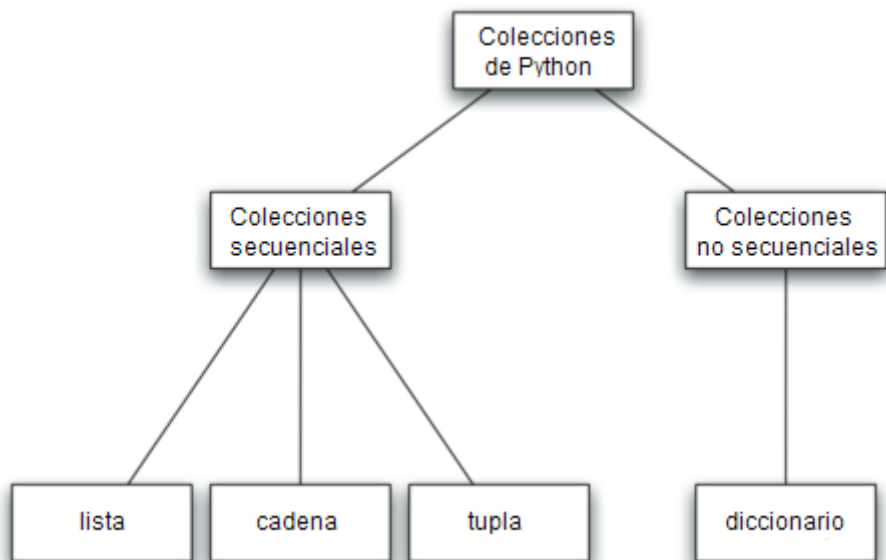
Out[11]: 9

```
In [12]: a, b = 2, 5  
        a + 2 * b
```

Out[12]: 12

## 5-Lenguaje python

# COLECCIONES



# 5-Lenguaje python

## Listas

La lista es un tipo de colección ordenada. Sería equivalente a lo que en otros lenguajes se conoce por arrays, o vectores.

Las listas pueden contener cualquier tipo de dato: números, cadenas, booleanos, ... y también listas.

Crear una lista es tan sencillo como indicar entre corchetes, y separados por comas, los valores que queremos incluir en la lista:

```
l = [22, True, "una lista", [1, 2]]
```

# 5-Lenguaje python - Listas

Podemos acceder a cada uno de los elementos de la lista escribiendo el nombre de la lista e indicando el índice del elemento entre corchetes. Ten en cuenta sin embargo que el índice del primer elemento de la lista es 0, y no 1:

```
l = [11, False]
mi_var = l[0] # mi_var vale 11
```

Si queremos acceder a un elemento de una lista incluida dentro de otra lista tendremos que utilizar dos veces este operador, primero para indicar a qué posición de la lista exterior queremos acceder, y el segundo para seleccionar el elemento de la lista interior:

```
l = ["una lista", [1, 2]]
mi_var = l[1][0] # mi_var vale 1
```

# 5-Lenguaje python - Listas

También podemos utilizar este operador para modificar un elemento de la lista si lo colocamos en la parte izquierda de una asignación:

```
l = [22, True]
l[0] = 99 # Con esto l valdrá [99, True]
```

Una curiosidad sobre el operador `[]` de Python es que podemos utilizar también números negativos. Si se utiliza un número negativo como índice, esto se traduce en que el índice empieza a contar desde el final, hacia la izquierda; es decir, con `[-1]` accederíamos al último elemento de la lista, con `[-2]` al penúltimo, con `[-3]`, al antepenúltimo, y así sucesivamente.



# 5-Lenguaje python - Listas

Otra cosa inusual es lo que en Python se conoce como *slicing* o particionado, y que consiste en ampliar este mecanismo para permitir seleccionar porciones de la lista. Si en lugar de un número escribimos dos números inicio y fin separados por dos puntos (inicio:fin) Python interpretará que queremos una lista que vaya desde la posición inicio a la posición fin, sin incluir este último. Si escribimos tres números (inicio:fin:salto) en lugar de dos, el tercero se utiliza para determinar cada cuantas posiciones añadir un elemento a la lista.

```
l = [99, True, "una lista", [1, 2]]  
mi_var = l[0:2]    # mi_var vale [99, True]  
mi_var = l[0:4:2]  # mi_var vale [99, "una lista"]
```

# 5-Lenguaje python - Listas

Hay que mencionar así mismo que no es necesario indicar el principio y el final del slicing, sino que, si estos se omiten, se usarán por defecto las posiciones de inicio y fin de la lista, respectivamente:

```
l = [99, True, "una lista"]
mi_var = l[1:] # mi_var vale [True, "una lista"]
mi_var = l[:2] # mi_var vale [99, True]
mi_var = l[:] # mi_var vale [99, True, "una lista"]
mi_var = l[::2] # mi_var vale [99, "una lista"]
```

# 5-Lenguaje python - Listas

También podemos utilizar este mecanismo para modificar la lista:

```
l = [99, True, "una lista", [1, 2]]  
l[0:2] = [0, 1] # l vale [0, 1, "una lista", [1, 2]]
```

pudiendo incluso modificar el tamaño de la lista si la lista de la parte derecha de la asignación tiene un tamaño menor o mayor que el de la selección de la parte izquierda de la asignación:

```
l[0:2] = [False] # l vale [False, "una lista", [1, 2]]
```

# 5-Lenguaje python

## Tuplas

Todo lo que hemos explicado sobre las listas se aplica también a las tuplas, a excepción de la forma de definirla, para lo que se utilizan paréntesis en lugar de corchetes.

```
t = (1, 2, True, "python")
```

En realidad el constructor de la tupla es la coma, no el paréntesis, pero el intérprete muestra los paréntesis, y nosotros deberíamos utilizarlos, por claridad.

```
>>> t = 1, 2, 3
>>> type(t)
type "tuple"
```

# 5-Lenguaje python - Tuplas

Además hay que tener en cuenta que es necesario añadir una coma para tuplas de un solo elemento, para diferenciarlo de un elemento entre paréntesis.

```
>>> t = (1)
>>> type(t)
type "int"
>>> t = (1,)
>>> type(t)
type "tuple"
```

# 5-Lenguaje python - Tuplas

Para referirnos a elementos de una tupla, como en una lista, se usa el operador []:

```
mi_var = t[0] # mi_var es 1  
mi_var = t[0:2] # mi_var es (1, 2)
```

# 5-Lenguaje python - Tuplas

Podemos utilizar el operador `[]` debido a que las tuplas, al igual que las listas, forman parte de un tipo de objetos llamados secuencias.

Una cadena también  
es una secuencia...

```
c = "hola mundo"
c[0]    # h
c[5:]   # mundo
c[:3]   # hauo
```

# 5-Lenguaje python - Tuplas

- Las tuplas son inmutables: Una vez creadas no se puede hacer update sobre los datos asignados inicialmente
- Tienen un tamaño fijo
- Las tuplas son más “livianas” que las listas
- Se aconsejan solo para ahorrar memoria



# 5-Lenguaje python

## Diccionarios

Los diccionarios, también llamados matrices asociativas, deben su nombre a que son colecciones que relacionan una clave y un valor. Por ejemplo, veamos un diccionario de películas y directores:

```
d = {"Love Actually ": "Richard Curtis",  
     "Kill Bill": "Tarantino",  
     "Amélie": "Jean-Pierre Jeunet"}
```

# 5-Lenguaje python - Diccionarios

La diferencia principal entre los diccionarios y las listas o las tuplas es que a los valores almacenados en un diccionario se les accede no por su índice, porque de hecho no tienen orden, sino por su clave, utilizando de nuevo el operador `[]`.

```
d["Love Actually "] # devuelve "Richard Curtis"
```

Al igual que en listas y tuplas también se puede utilizar este operador para reasignar valores.


```
d["Kill Bill"] = "Quentin Tarantino"
```

# 5-Lenguaje python

## Definición de funciones

**Ejercicio 1.** Definir la función suma tal que suma(x, y) es la suma de x e y. . Por ejemplo,

```
>>> suma(2,3)
5
```



```
In [13]: def suma(x, y):
         return x+y
```

```
In [14]: suma(2, 3)
```

```
Out[14]: 5
```

**Aclaración importante:** Las instrucciones que pertenecen a un segmento del código (Condicionales, funciones, bucles, etc.) se tabulan con 4 espacios, no se aconseja la tecla tab.

# 5-Lenguaje python

## Escritura y lectura

**Ejercicio 2.** Definir el procedimiento suma que lea dos números y escriba su suma. Por ejemplo,

```
>>> suma()  
Escribe el primer número: 2  
Escribe el segundo número: 3  
La suma es: 5
```

Se aconseja el uso de cast directo a int en el caso de números enteros -> `int(input(...`

```
In [15]: def suma():  
         a = eval(input("Escribe el primer número: "))  
         b = eval(input("Escribe el segundo número: "))  
         print("La suma es:",a+b)
```

eval podría ejecutar código malicioso. Cuidado!

```
In [16]: suma()
```

```
La suma es: 5
```

# 5-Lenguaje python

## Condicionales simples

**Ejercicio 3.** Definir, usando condicionales, la función maximo tal que maximo(x,y) es el máximo de x e y. Por ejemplo,

```
>>> maximo(2, 5)
5
>>> maximo(2, 1)
2
```

# 5-Lenguaje python

```
In [17]: def maximo(x, y) :  
         if x > y:  
             return x  
         else:  
             return y
```

```
In [18]: maximo(2, 5)
```

```
Out[18]: 5
```

```
In [19]: maximo (2, 1)
```

```
Out[19]: 2
```

# 5-Lenguaje python

## Condicionales múltiples

**Ejercicio 4.** Definir la función signo tal que signo(x) es el signo de x. Por ejemplo,

```
>>> signo(5)
1
>>> signo(-7)
- 1
>>> signo(0)
0
```

# 5-Lenguaje python

```
In [20]: def signo(x):  
         if x > 0:  
             return 1  
         elif x < 0:  
             return -1  
         else:  
             return 0
```

```
In [21]: signo(5)
```

```
Out[21]: 1
```

```
In [22]: signo(-7)
```

```
Out[22]: -1
```

```
In [23]: signo(0)
```

```
Out[23]: 0
```



# 5-Lenguaje python

## Estructuras iterativas

### Bucles mientras

**Ejercicio 5.** Definir, con un bucle while, la función sumaImpares tal que sumaImpares(n) es la suma de los n primeros números impares. Por ejemplo,

```
>>> sumaImpares(3)
9
>>> sumaImpares(4)
16
```

# 5-Lenguaje python

```
In [24]: def sumaImpares(n):  
         s, k = 0, 0  
         while k < n:  
             s = s + 2*k + 1  
             k = k + 1  
         return s
```

```
In [25]: sumaImpares(3)
```

```
Out[25]: 9
```

```
In [26]: sumaImpares(4)
```

```
Out[26]: 16
```

# 5-Lenguaje python

**Ejercicio 6.** Definir la función mayorExponente tal que mayorExponente(a,n) es el mayor k tal que  $a^k$  divide a n. Por ejemplo,

```
>>> mayorExponente(2,40);  
3
```

```
In [27]: def mayorExponente(a, n):  
         k = 0  
         while (n % a == 0):  
             n = n/a  
             k = k + 1  
         return k
```

```
In [28]: mayorExponente(2, 40)
```

```
Out[28]: 3
```

# 5-Lenguaje python

## Bucle para

**Ejercicio 7.** Definir, por iteración con for, la función fact tal que fact(n) es el factorial de n. Por ejemplo,

```
>>> fact 4
24
```

Sentencia for range (start (opcional) = 0, stop, step (opcional) = 1)

```
In [29]: def fact(n):
          f = 1
          for k in range(1,n+1):
              f = f * k
          return f
```

```
In [30]: fact(4)
```

```
Out[30]: 24
```

# 5-Lenguaje python

## Bucle para sobre listas

**Ejercicio 8.** Definir, por iteración, la función suma tal que suma(xs) es a suma de los números de la lista xs. Por ejemplo,

```
>>> suma([3,2,5])  
10
```

```
In [31]: def suma(xs):  
         r = 0  
         for x in xs:  
             r = x + r  
         return r
```

```
In [32]: suma([3, 2, 5])
```

```
Out[32]: 10
```

# 5-Lenguaje python

## Recursión

**Ejercicio 9.** Definir, por recursión, la función fact tal que factR(n) es el factorial de n. Por ejemplo,

```
>>> fact 4
24
```

```
In [33]: def fact(n):
        if n == 0:
            return 1
        else:
            return n * fact(n-1)
```

```
In [34]: fact(4)
```

```
Out[34]: 24
```

# 5-Lenguaje python

## La función de Takeuchi

```
In [35]: def tak(x,y,z):  
         if x <= y:  
             return y  
         else:  
             return tak(tak(x - 1, y, z),  
                         tak(y - 1, z, x),  
                         tak(z - 1, x, y))
```

Importación de la librería para medir el tiempo.

```
In [36]: import time
```

# 5-Lenguaje python

```
#Probamos...

t_inicial = time.time()
tak(11,6,1)
t_final = time.time() - t_inicial
print(f"Tiempo {t_final} en segundos")

tak(12,6,1)
tak(13,6,1)
tak(14,6,1)
```



# 5-Lenguaje python

## Documentación (Docstring) y tipado estático de parámetros

```
def sumar(num1:int, num2:int) -> int:
    """Función que retorna la suma de dos enteros...
    num1: Sumando 1
    num2: Sumando 2
    retorno: Total de tipo entero
    """
    return num1 + num2

print("La suma calculada es: ", sumar(4,6))
```

```
def sumar(num1: int, num2: int)
Función que retorna la suma de dos enteros... num1: Sumando 1 num2: Sumando 2 retorno: Total de tipo entero
```

# 5-Lenguaje python

## Algunos detalles más para las funciones...

Los valores por defecto para los parámetros se definen situando un signo igual después del nombre del parámetro y a continuación el valor por defecto:

```
def imprimir(texto, veces = 1):  
    print veces * texto
```

En el ejemplo anterior si no indicamos un valor para el segundo parámetro se imprimirá una sola vez la cadena que le pasamos como primer parámetro:

```
>>> imprimir("hola")  
hola
```

si se le indica otro valor, será este el que se utilice:

```
>>> imprimir("hola", 2)  
holahola
```

# 5-Lenguaje python

## Algunos detalles más para las funciones...

Para definir funciones con un número variable de argumentos colocamos un último parámetro para la función cuyo nombre debe precederse de un signo \*:

```
def varios(param1, param2, *otros):  
    for val in otros:  
        print val
```

```
varios(1, 2)  
varios(1, 2, 3)  
varios(1, 2, 3, 4)
```

# 5-Lenguaje python

## Algunos detalles más para las funciones...

En el siguiente ejemplo se utiliza la función `items` de los diccionarios, que devuelve una lista con sus elementos, para imprimir los parámetros que contiene el diccionario.

```
def varios(param1, param2, **otros):  
    for i in otros.items():  
        print i  
  
varios(1, 2, tercero = 3)
```

# 5-Lenguaje python

## Algunos detalles más para las funciones...

```
def f(x, y):  
    x = x + 3  
    y.append(23)  
    print x, y
```

```
x = 22  
y = [22]  
f(x, y)  
print x, y
```

El resultado de la ejecución de este programa sería

```
25 [22, 23]  
22 [22, 23]
```

¿Qué explicación le damos a este código?

# 5-Lenguaje python



Funciones útiles de Listas, diccionarios y cadenas

# 5-Lenguaje python

## Diccionarios

- `D.get(k[, d])` Busca el valor de la clave `k` en el diccionario. Es equivalente a utilizar `D[k]` pero al utilizar este método podemos indicar un valor a devolver por defecto si no se encuentra la clave, mientras que con la sintaxis `D[k]`, de no existir la clave se lanzaría una excepción.

# 5-Lenguaje python

```
print(diccionario.has_key("clave"))  
# Reemplazada en python 3 por:  
print('clave' in diccionario)
```

## Diccionarios



- **D.has\_key(k)** Comprueba si el diccionario tiene la clave k. Es equivalente a la sintaxis `k in D`.
- **D.items()** Devuelve una lista de tuplas con pares clave-valor
- **D.keys()** Devuelve una lista de las claves del diccionario.



# 5-Lenguaje python

## Diccionarios

- `D.pop(k[, d])` Borra la clave `k` del diccionario y devuelve su valor. Si no se encuentra dicha clave se devuelve `d` si se especificó el parámetro o bien se lanza una excepción.
- `D.values()` Devuelve una lista de los valores del diccionario.

# 5-Lenguaje python

## Cadenas

- `S.count(sub[, start[, end]])` Devuelve el número de veces que se encuentra `sub` en la cadena. Los parámetros opcionales `start` y `end` definen una subcadena en la que buscar.
- `S.find(sub[, start[, end]])` Devuelve la posición en la que se encontró por primera vez `sub` en la cadena o `-1` si no se encontró.

# 5-Lenguaje python

## Cadenas

- `S.join(sequence)` Devuelve una cadena resultante de concatenar las cadenas de la secuencia `seq` separadas por la cadena sobre la que se llama el método.
- `S.partition(sep)` Busca el separador `sep` en la cadena y devuelve una tupla con la subcadena hasta dicho separador, el separador en si, y la subcadena del separador hasta el final de la cadena. Si no se encuentra el separador, la tupla contendrá la cadena en si y dos cadenas vacías.

# 5-Lenguaje python

## Cadenas

- `S.replace(old, new[, count])` Devuelve una cadena en la que se han reemplazado todas las ocurrencias de la cadena `old` por la cadena `new`. Si se especifica el parámetro `count`, este indica el número máximo de ocurrencias a reemplazar.
- `S.split([sep [,maxsplit]])` Devuelve una lista conteniendo las subcadenas en las que se divide nuestra cadena al dividirlas por el delimitador `sep`. En el caso de que Revisitando objetos 55 no se especifique `sep`, se usan espacios. Si se especifica `maxsplit`, este indica el número máximo de particiones a realizar.

# 5-Lenguaje python

## Listas

- `L.append(object)` Añade un objeto al final de la lista.  
`L.count(value)` Devuelve el número de veces que se encontró `value` en la lista.
- `L.extend(iterable)` Añade los elementos del iterable a la lista.
- `L.index(value[, start[, stop]])` Devuelve la posición en la que se encontró la primera ocurrencia de `value`. Si se especifican, `start` y `stop` definen las posiciones de inicio y fin de una sublista en la que buscar.

# 5-Lenguaje python

## Listas

- `L.insert(index, object)` Inserta el objeto `object` en la posición `index`.
- `L.pop([index])` Devuelve el valor en la posición `index` y lo elimina de la lista. Si no se especifica la posición, se utiliza el último elemento de la lista.
- `L.remove(value)` Eliminar la primera ocurrencia de `value` en la lista

# 5-Lenguaje python

## Listas

- `L.reverse()` Invierte la lista. Esta función trabaja sobre la propia lista desde la que se invoca el método, no sobre una copia.
- `L.sort(cmp=None, key=None, reverse=False)` Ordena la lista. Si se especifica `cmp`, este debe ser una función que tome como parámetro dos valores `x` e `y` de la lista y devuelva `-1` si `x` es menor que `y`, `0` si son iguales y `1` si `x` es mayor que `y`. El parámetro `reverse` es un booleano que indica si se debe ordenar la lista de forma inversa, lo que sería equivalente a llamar primero a `L.sort()` y después a `L.reverse()`.

